

## HW8 : lab4 실습

20160394 임효상

### 1) 실습 A

The screenshot shows a debugger interface with the following components:

- Assembly Code (Left):**

```

.global matrix
matrix:
5: stmf sp!, [r0-r12,lr]
6:
7:   mov r6,#0
8:   mov r7,#0
9:   mov r8,#0
10:  mov r9,#0
11:
12: Loop:
13:   mov r11, #12
14:   mul r11,r6,r11
15:   add r11,r11,r7,LSL #2
16:   add r11,r11,r0
17:
19:   ldr r12,[r11]
20:
21:   mov r11, #12
22:   mul r11,r7,r11
23:   add r11,r11,r8,LSL #2
24:   add r11,r11,r1
25:
26:   ldr r11,[r11]
27:
28:   mul r12,r11,r12
29:   add r9, r9, r12
30:
31:   add r7,r7,#1
32:   cmp r7,r3
33:   bne Loop
34:   mov r7,#0

```
- Memory Dump (Right):**

Address	Value (hex)	Value (decimal)	Description
0xfffff0ec	04 00 00 00 05 00 00 00	06 00 00 00	.....
0xfffff0f8	07 00 00 00 08 00 00 00	09 00 00 00	.....
0xfffff104	09 00 00 00 08 00 00 00	07 00 00 00	.....
0xfffff110	06 00 00 00 05 00 00 00	04 00 00 00	.....
0xfffff11c	03 00 00 00 02 00 00 00	01 00 00 00	.....
0xfffff128	1e 00 00 00 18 00 00 00	12 00 00 00	.....
0xfffff134	54 00 00 00 45 00 00 00	58 01 01 00	T...E...X...
0xfffff140	08 b4 08 00 58 01 01 00	00 00 00 00	....X.....
0xfffff14c	20 9b 08 00 00 00 00 00		.....
- Registers (Right):**

name	value (hex)	value (decimal)	description
r0	0xfffff0e0	4294897888	
r1	0xfffff104	4294897924	
r2	0xfffff128	4294897960	
r3	0x3	3	
r4	0x10a8	69800	
r5	0x10158	65880	
r6	0x1	1	
r7	0x0	0	
r8	0x2	2	register 8 (64-bit)

r6, r7, r8이 각각 1,0,2가 된 시점이다. 현재 배열 C는 [0][0]~[1][1]번방까지 초기화된 상태이다. 다음 계산은 C[1][2]를 계산할 차례이다.

The screenshot shows a debugger interface with the following components:

- Assembly Code (Left):**

```

.global matrix
matrix:
5: stmf sp!, [r0-r12,lr]
6:
7:   mov r6,#0
8:   mov r7,#0
9:   mov r8,#0
10:  mov r9,#0
11:
12: Loop:
13:   mov r11, #12
14:   mul r11,r6,r11
15:   add r11,r11,r7,LSL #2
16:   add r11,r11,r0
17:
19:   ldr r12,[r11]
20:
21:   mov r11, #12
22:   mul r11,r7,r11
23:   add r11,r11,r8,LSL #2
24:   add r11,r11,r1
25:
26:   ldr r11,[r11]
27:
28:   mul r12,r11,r12
29:   add r9, r9, r12
30:
31:   add r7,r7,#1
32:   cmp r7,r3
33:   bne Loop
34:   mov r7,#0

```
- Memory Dump (Right):**

Address	Value (hex)	Value (decimal)	Description
0xfffff0e0	01 00 00 00 02 00 00 00	03 00 00 00	.....
0xfffff0ec	04 00 00 00 05 00 00 00	06 00 00 00	.....
0xfffff0f8	07 00 00 00 08 00 00 00	09 00 00 00	.....
0xfffff104	09 00 00 00 08 00 00 00	07 00 00 00	.....
0xfffff110	06 00 00 00 05 00 00 00	04 00 00 00	.....
0xfffff11c	03 00 00 00 02 00 00 00	01 00 00 00	.....
0xfffff128	1e 00 00 00 18 00 00 00	12 00 00 00	.....
0xfffff134	54 00 00 00 45 00 00 00	58 01 01 00	T...E...X...
0xfffff140	08 b4 08 00 58 01 01 00	00 00 00 00	....X.....
0xfffff14c	20 9b 08 00 00 00 00 00		.....
- Registers (Right):**

name	value (hex)	value (decimal)	description
r0	0xfffff0e0	4294897888	
r1	0xfffff104	4294897924	
r2	0xfffff128	4294897960	
r3	0x3	3	
r4	0x10a8	69800	
r5	0x10158	65880	
r6	0x1	1	
r7	0x0	0	
r8	0x2	2	register 8 (64-bit)
r9	0x0	0	register 9 (64-bit)
r10	0x89b04	563972	register 10 (64-bit)
r11	0xfffff0ec	4294897900	register 11 (64-bit)
r12	0x4	4	register 12 (64-bit)

**LDR r12, [r11] :** 배열 A에서 값(p.20그림의 1번값)을 가져왔다. r11에 저장된 메모리 주소는 A[1][0]의 주소이다. 이에 따라 A[1][0]의 값이 4이므로 r12가 4가 되었다.

```

34      mov     r7,#0
35
36
37      mov     r11, #12
38

```

r10	0x89b04	563972	register 10 (64-bit)
r11	0xffff10c	4294897932	register 11 (64-bit)
r12	0x4	4	register 12 (64-bit)

show filesystem

fetch disassembly

reload file

jump to line

/home/bbo/CA\_2021/lab4/matrix.s:28 (62 lines total)

NI

SI

```

2      .global matrix
3
4      matrix:
5      stmfd sp!, {r0-r12,lr}
6
7      mov     r6,#0
8      mov     r7,#0
9      mov     r8,#0
10     mov     r9,#0
11
12     Loop:
13     mov     r11, #12
14     mul     r11,r6,r11
15     add     r11,r11,r7,LSL #2
16
17     add     r11,r11,r0
18
19     ldr     r12,[r11]
20
21     mov     r11, #12
22     mul     r11,r7,r11
23     add     r11,r11,r8,LSL #2
24     add     r11,r11,r1
25
26     ldr     r11,[r11]
27
28     mul     r12,r11,r12
29     add     r9, r9, r12
30
31     add     r7,r7,#1
32     cmp     r7,r3
33     bne     Loop
34     mov     r7,#0
35
36
37     mov     r11, #12
38     mul     r11,r6,r11
39     add     r11,r11,r8,LSL #2

```

0xffff0e0

01 00 00 00 02 00 00 00 03 00 00 00

0xffff0ec

04 00 00 00 05 00 00 00 06 00 00 00

0xffff0f8

07 00 00 00 08 00 00 00 09 00 00 00

0xffff104

09 00 00 00 08 00 00 00 07 00 00 00

0xffff110

06 00 00 00 05 00 00 00 04 00 00 00

0xffff11c

03 00 00 00 02 00 00 00 01 00 00 00

0xffff128

1e 00 00 00 18 00 00 00 12 00 00 00

0xffff134

54 00 00 00 45 00 00 00 58 01 01 00

0xffff140

08 b4 08 00 58 01 01 00 00 00 00 00

0xffff14c

20 9b 08 00 00 00 00 00 00

> breakpoints

> signals

▼ registers

name	value (hex)	value (decimal)	description
r0	0xffff0e0	4294897888	
r1	0xffff104	4294897924	
r2	0xffff128	4294897960	
r3	0x3	3	
r4	0x110a8	69800	
r5	0x10158	65880	
r6	0x1	1	
r7	0x0	0	
r8	0x2	2	register 8 (64-bit)
r9	0x0	0	register 9 (64-bit)
r10	0x89b04	563972	register 10 (64-bit)
r11	0x7	7	register 11 (64-bit)
r12	0x4	4	register 12 (64-bit)

**LDR r11, [r11]** : 배열 B에서 값(p.20그림의 2번값)을 가져왔다. r11에 저장된 메모리 주소는 B[0][2]의 주소였다. 이에 따라 B[0][2]의 값이 7이므로 r11가 7이 되었다.

```

1      .text
2      .global matrix
3
4      matrix:
5      stmfd sp!, {r0-r12,lr}
6
7      mov     r6,#0
8      mov     r7,#0
9      mov     r8,#0
10     mov     r9,#0
11
12     Loop:
13     mov     r11, #12
14     mul     r11,r6,r11
15     add     r11,r11,r7,LSL #2
16
17     add     r11,r11,r0
18
19     ldr     r12,[r11]
20
21     mov     r11, #12
22     mul     r11,r7,r11
23     add     r11,r11,r8,LSL #2
24     add     r11,r11,r1
25
26     ldr     r11,[r11]
27
28     mul     r12,r11,r12
29     add     r9, r9, r12
30
31     add     r7,r7,#1
32     cmp     r7,r3
33     bne     Loop
34     mov     r7,#0
35
36
37     mov     r11, #12
38     mul     r11,r6,r11
39     add     r11,r11,r8,LSL #2

```

0xffff0e0

01 00 00 00 02 00 00 00 03 00 00 00

0xffff0ec

04 00 00 00 05 00 00 00 06 00 00 00

0xffff0f8

07 00 00 00 08 00 00 00 09 00 00 00

0xffff104

09 00 00 00 08 00 00 00 07 00 00 00

0xffff110

06 00 00 00 05 00 00 00 04 00 00 00

0xffff11c

03 00 00 00 02 00 00 00 01 00 00 00

0xffff128

1e 00 00 00 18 00 00 00 12 00 00 00

0xffff134

54 00 00 00 45 00 00 00 58 01 01 00

0xffff140

08 b4 08 00 58 01 01 00 00 00 00 00

> breakpoints

> signals

▼ registers

name	value (hex)	value (decimal)	description
r0	0xffff0e0	4294897888	
r1	0xffff104	4294897924	
r2	0xffff128	4294897960	
r3	0x3	3	
r4	0x110a8	69800	
r5	0x10158	65880	
r6	0x1	1	
r7	0x0	0	
r8	0x2	2	register 8 (64-bit)
r9	0x1c	28	register 9 (64-bit)
r10	0x89b04	563972	register 10 (64-bit)
r11	0x7	7	register 11 (64-bit)
r12	0x1c	28	register 12 (64-bit)
sp	0xffff098	4294897816	

두 값 r11=7 과 r12=4 를 곱한 28 의 결과값을 r12 에 저장한 후 r9 에 더한다. r9 는 0 이였으므로 이제 r9 = 28 이다.

The screenshot shows a debugger interface with the following assembly code on the left:

```

1 .text
2 .global matrix
3
4 matrix:
5     stmfid sp!, [r0-r12,lr]
6
7     mov     r6,#0
8     mov     r7,#0
9     mov     r8,#0
10    mov     r9,#0
11
12 Loop:
13     mov     r11, #12
14     mul     r11,r6,r11
15     add     r11,r11,r7,LSL #2
16
17     add     r11,r11,r0
18
19     ldr     r12,[r11]
20
21     mov     r11, #12
22     mul     r11,r7,r11
23     add     r11,r11,r8,LSL #2
24     add     r11,r11,r1
25
26     ldr     r11,[r11]
27
28     mul     r12,r11,r12
29     add     r9, r9, r12
30
31     add     r7,r7,#1
32     cmp     r7,r3
33     bne     Loop
34     mov     r7,#0
35
36     mov     r11, #12
37     mul     r11,r6,r11

```

On the right, the register window shows the following values:

name	value (hex)	value (decimal)	description
r0	0xffff0e0	4294897888	
r1	0xffff104	4294897924	
r2	0xffff128	4294897960	
r3	0x3	3	
r4	0x110a8	69800	
r5	0x10158	65880	
r6	0x1	1	
<b>r7</b>	<b>0x1</b>	<b>1</b>	
r8	0x2	2	register 8 (64-bit)
r9	0x1c	28	register 9 (64-bit)
r10	0x89b04	563972	register 10 (64-bit)
r11	0x7	7	register 11 (64-bit)
r12	0x1c	28	register 12 (64-bit)
sp	0xffff098	4294897816	

다음 연산  $A[1][1] * B[1][2]$ 를 하기 위해 r7 값이 1 만큼 더해진다.

The screenshot shows a debugger interface with the following assembly code on the left:

```

1 .text
2 .global matrix
3
4 matrix:
5     stmfid sp!, [r0-r12,lr]
6
7     mov     r6,#0
8     mov     r7,#0
9     mov     r8,#0
10    mov     r9,#0
11
12 Loop:
13     mov     r11, #12
14     mul     r11,r6,r11
15     add     r11,r11,r7,LSL #2
16
17     add     r11,r11,r0
18
19     ldr     r12,[r11]
20
21     mov     r11, #12
22     mul     r11,r7,r11
23     add     r11,r11,r8,LSL #2
24     add     r11,r11,r1
25
26     ldr     r11,[r11]
27
28     mul     r12,r11,r12
29     add     r9, r9, r12
30
31     add     r7,r7,#1
32     cmp     r7,r3
33     bne     Loop
34     mov     r7,#0
35
36     mov     r11, #12
37     mul     r11,r6,r11

```

On the right, the register window shows the following values:

name	value (hex)	value (decimal)	description
r0	0xffff0e0	4294897888	
r1	0xffff104	4294897924	
r2	0xffff128	4294897960	
r3	0x3	3	
r4	0x110a8	69800	
r5	0x10158	65880	
r6	0x1	1	
r7	0x1	1	
r8	0x2	2	register 8 (64-bit)
r9	0x1c	28	register 9 (64-bit)
r10	0x89b04	563972	register 10 (64-bit)
<b>r11</b>	<b>0xffff0f0</b>	<b>4294897904</b>	register 11 (64-bit)
<b>r12</b>	<b>0x5</b>	<b>5</b>	register 12 (64-bit)
sp	0xffff098	4294897816	

**LDR r12, [r11]** : 배열 A에서 값(p.20그림의 1번값)을 가져왔다. r11에 저장된 메모리 주소는  $A[1][1]$ 의 주소이다. 이에 따라  $A[1][1]$ 의 값이 5이므로 r12가 5가 되었다.

The screenshot shows a debugger window with assembly code on the left and a register window on the right. The assembly code includes instructions for moving, multiplying, and adding registers. The register window shows the current values of registers r0 through r12 and the stack pointer (sp). Register r11 is highlighted with a red box, showing a value of 0x4 (decimal 4).

name	value (hex)	value (decimal)	description
r0	0xffff0e0	4294897888	
r1	0xffff104	4294897924	
r2	0xffff128	4294897960	
r3	0x3	3	
r4	0x110a8	69800	
r5	0x10158	65880	
r6	0x1	1	
r7	0x1	1	
r8	0x2	2	register 8 (64-bit)
r9	0x1c	28	register 9 (64-bit)
r10	0x89b04	563972	register 10 (64-bit)
r11	0x4	4	register 11 (64-bit)
r12	0x5	5	register 12 (64-bit)
sp	0xffff098	4294897816	

**LDR r11, [r11]** : 배열 B에서 값(p.20그림의 2번값)을 가져왔다. r11에 저장된 메모리 주소는 B[1][2]의 주소였다. 이에 따라 B[1][2]의 값이 4이므로 r11가 4가 되었다.

The screenshot shows a debugger window with assembly code on the left and a register window on the right. The assembly code includes instructions for moving, multiplying, and adding registers. The register window shows the current values of registers r0 through r12 and the stack pointer (sp). Register r9 is highlighted with a red box, showing a value of 0x38 (decimal 48). Register r11 is also highlighted with a red box, showing a value of 0x4 (decimal 4). Register r12 is highlighted with a red box, showing a value of 0x14 (decimal 20).

name	value (hex)	value (decimal)	description
r0	0xffff0e0	4294897888	
r1	0xffff104	4294897924	
r2	0xffff128	4294897960	
r3	0x3	3	
r4	0x110a8	69800	
r5	0x10158	65880	
r6	0x1	1	
r7	0x1	1	
r8	0x2	2	register 8 (64-bit)
r9	0x38	48	register 9 (64-bit)
r10	0x89b04	563972	register 10 (64-bit)
r11	0x4	4	register 11 (64-bit)
r12	0x14	20	register 12 (64-bit)
sp	0xffff098	4294897816	

두 값 r11=4 과 r12=5 를 곱한 20 의 결과값을 r12 에 저장한 후 r9 에 더한다. r9 는 28 이었으므로 이제 r9 = 48 이다.

Assembly code (lines 1-33):

```

1  mov r6,#0
2  mov r7,#0
3  mov r8,#0
4  mov r9,#0
5
6  Loop:
7  mov r11,#12
8  mul r11,r6,r11
9  add r11,r11,r7,LSL #2
10
11  add r11,r11,r0
12
13  ldr r12,[r11]
14
15  mov r11,#12
16  mul r11,r7,r11
17  add r11,r11,r8,LSL #2
18  add r11,r11,r1
19
20  ldr r11,[r11]
21
22  mul r12,r11,r12
23  add r9,r9,r12
24
25  add r7,r7,#1
26  cmp r7,r3
27  bne Loop
28  mov r7,#0
29
30  mov r11,#12
31  mul r11,r6,r11
32  add r11,r11,r8,LSL #2
33  add r11,r11,r2

```

Registers (r0-r12, sp):

name	value (hex)	value (decimal)	description
r0	0xffff0e0	4294897888	
r1	0xffff104	4294897924	
r2	0xffff128	4294897960	
r3	0x3	3	
r4	0x110a8	69800	
r5	0x10158	65880	
r6	0x1	1	
r7	0x2	2	
r8	0x2	2	register 8 (64-bit)
r9	0x30	48	register 9 (64-bit)
r10	0x89b04	563972	register 10 (64-bit)
r11	0x4	4	register 11 (64-bit)
r12	0x14	20	register 12 (64-bit)
sp	0xffff098	4294897816	

다음 연산  $A[1][2] \times B[2][2]$ 를 하기 위해 r7 값이 1 만큼 더해진다.

Assembly code (lines 1-33):

```

1  mov r6,#0
2  mov r7,#0
3  mov r8,#0
4  mov r9,#0
5
6  Loop:
7  mov r11,#12
8  mul r11,r6,r11
9  add r11,r11,r7,LSL #2
10
11  add r11,r11,r0
12
13  ldr r12,[r11]
14
15  mov r11,#12
16  mul r11,r7,r11
17  add r11,r11,r8,LSL #2
18  add r11,r11,r1
19
20  ldr r11,[r11]
21
22  mul r12,r11,r12
23  add r9,r9,r12
24
25  add r7,r7,#1
26  cmp r7,r3
27  bne Loop
28  mov r7,#0
29
30  mov r11,#12
31  mul r11,r6,r11
32  add r11,r11,r8,LSL #2
33  add r11,r11,r2

```

Registers (r0-r12, sp):

name	value (hex)	value (decimal)	description
r0	0xffff0e0	4294897888	
r1	0xffff104	4294897924	
r2	0xffff128	4294897960	
r3	0x3	3	
r4	0x110a8	69800	
r5	0x10158	65880	
r6	0x1	1	
r7	0x2	2	
r8	0x2	2	register 8 (64-bit)
r9	0x30	48	register 9 (64-bit)
r10	0x89b04	563972	register 10 (64-bit)
r11	0xffff0f4	4294897908	register 11 (64-bit)
r12	0x4	4	register 12 (64-bit)
sp	0xffff098	4294897816	

**LDR r12, [r11]** : 배열 A에서 값(p.20그림의 1번값)을 가져왔다. r11에 저장된 메모리 주소는  $A[1][2]$ 의 주소이다. 이에 따라  $A[1][2]$ 의 값이 6이므로 r12가 6이 되었다.

The screenshot shows a debugger interface with assembly code on the left and a register window on the right. The assembly code includes instructions like `mov r11, #12`, `mul r11, r6, r11`, and `add r11, r11, r8, LSL #2`. The register window shows the current values of registers r0 through r12 and the stack pointer (sp). Register r11 is highlighted with a red box, showing a value of 0x1.

name	value (hex)	value (decimal)	description
r0	0xfffff0e0	4294897888	
r1	0xfffff104	4294897924	
r2	0xfffff128	4294897960	
r3	0x3	3	
r4	0x110a8	69800	
r5	0x10158	65880	
r6	0x1	1	
r7	0x2	2	
r8	0x2	2	register 8 (64-bit)
r9	0x30	48	register 9 (64-bit)
r10	0x89b04	563972	register 10 (64-bit)
r11	0x1	1	register 11 (64-bit)
r12	0x6	6	register 12 (64-bit)
sp	0xfffff098	4294897816	

**LDR r11, [r11]** : 배열 B에서 값(p.20그림의 2번값)을 가져왔다. r11에 저장된 메모리 주소는 B[2][2]의 주소였다. 이에 따라 B[2][2]의 값이 4이므로 r11가 1이 되었다.

The screenshot shows a debugger interface with assembly code on the left and a register window on the right. The assembly code includes instructions like `mov r11, #12`, `mul r11, r6, r11`, and `add r11, r11, r8, LSL #2`. The register window shows the current values of registers r0 through r12 and the stack pointer (sp). Register r11 is highlighted with a red box, showing a value of 0x1. Register r12 is also highlighted with a red box, showing a value of 0x6.

name	value (hex)	value (decimal)	description
r0	0xfffff0e0	4294897888	
r1	0xfffff104	4294897924	
r2	0xfffff128	4294897960	
r3	0x3	3	
r4	0x110a8	69800	
r5	0x10158	65880	
r6	0x1	1	
r7	0x2	2	
r8	0x2	2	register 8 (64-bit)
r9	0x30	48	register 9 (64-bit)
r10	0x89b04	563972	register 10 (64-bit)
r11	0x1	1	register 11 (64-bit)
r12	0x6	6	register 12 (64-bit)
sp	0xfffff098	4294897816	

두 값 r11=1 과 r12=6 를 곱한 6 의 결과값을 r12 에 저장한 후 r9 에 더한다. r9 는 48 이었으므로 이제 r9 = 54 이다. 3 번값의 연산이 종료되었고 결과값으로 54 가 C[1][2]에 들어가게된다.

## 2) 실습 B

s,u,v를 3으로 입력 후:

```
bbo@ubuntu:~/CA_2021/lab4$ qemu-arm-static -g 8080 ./lab4
Input values of s,u,v : 333
Input value of A : 123456789
Input value of B : 123456789
Array A
1 2 3
4 5 6
7 8 9
Array B
1 2 3
4 5 6
7 8 9
Array C after Operating :
30 36 42
66 81 96
102 126 150
```

s=3,u=4,v=5로 입력

```
bbo@ubuntu:~/CA_2021/lab4$ qemu-arm-static -g 8080 ./lab4
Input values of s,u,v : 345
Input value of A : 123456789123
Input value of B : 12345678912345678912
Array A
1 2 3 4
5 6 7 8
9 1 2 3
Array B
1 2 3 4 5
6 7 8 9 1
2 3 4 5 6
7 8 9 1 2
Array C after Operating :
47 57 67 41 33
111 137 163 117 89
40 55 70 58 64
```